Efficient and Dynamic Response to Fire

Christian J. Darken

MOVES Institute and
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943
cjdarken@nps.edu

David J. Morgan

Operations Group,
National Training Center
U.S. Army
Fort Irwin, CA 92310-5107
david.j.morgan@us.army.mil

Gregory Paull

Secret Level Inc.
870 Market St., Suite 1015
San Francisco, CA 94102
greg@secretlevel.com

Abstract

Most first-person shooter game AI's are poor at quickly getting out of lines of fire. AI agents that pass up obvious opportunities to hide or take cover can ruin a game's immersiveness. We will present the sensor grid algorithm that can be used to address this problem, and has been implemented on top of America's Army. The algorithm performs a focused run-time search in the immediate vicinity of the agent. This allows it to be both fast and to react to changes in the environment.

Background

Taking cover is a universal human response to threat. However, it is not innate; children must learn to hide. It is also not totally understood; psychologists are still investigating a critical part of hiding, which is what we know of what other people can or cannot see (Kelly et. al.). Nonetheless, nearly everyone is able to quickly and effectively duck to safety when threatened. The use of cover is also not purely defensive in nature. A person can be taught to take advantage of cover when moving to make invisible shifts in their position and to minimize their exposure to danger when shooting.

The ability to use cover effectively is one of the skills that separate the best real players in first-person shooters from the average players. Unfortunately in the current state of gaming it is also one of the ways to distinguish between live players and game agents. Game agents do not use cover effectively. Typical problems include running right by good sources of cover, and failing to consistently take the most direct route to safety.

This paper describes an algorithm that allows software agents to perceive and exploit opportunities to duck out of the sight of a set of "observers" (generally, the hostile forces that might shoot at the agent). The goal is realistic short-term "react to fire" behavior, rather than constructing a concealed path to a specific goal. Strictly speaking, the algorithm as described finds concealment rather than cover, but extension to approximate cover finding is discussed. The approach performs a focused dynamic (i.e. run-time) search in the immediate vicinity of the agent. This allows it to be both fast and to react to changes in the geometry of the environment that occur during play. We first describe some related techniques already in the literature. Then, we describe the algorithm, first at a high level and then in detail. Next, we describe some details of our specific implementation on top of America's Army. We conclude with some ideas for optimizing and extending the algorithm. We have previously described some aspects of the sensor grid algorithm (Morgan 2003)(Darken 2004).

Related Work

Previous approaches to the hiding problem involve searching a fixed set of potential hiding locations. Often this is the set of waypoints used to plot movement.

Typically, navigation is accomplished in high-resolution shooter games by the use of a set of locations we call "waypoints". An agent gets from point A to point B by moving from A to a nearby waypoint. Then the agent moves from waypoint to waypoint until a waypoint close to B is reached. The waypoint set may be selected by hand, as is typical of games based on the Unreal engine, or the set may be selected by various algorithms (Stout 2000)(Snook 2000). It was early recognized that one key to keeping the computational requirements of searching the waypoint set manageable was to keep it as small as possible (Rabin 2000). Since waypoint infrastructure is so commonly available, it seems only natural to reuse it for determining places to hide (Reece 2003)(Reece 2000)(van der Sterren 2002).

The primary advantage of waypoint-based techniques is ease of implementation and low run-time computational complexity. Unfortunately, the latter benefit is only gained when the set of waypoints is small, and when it is small, the likelihood that the best

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
2004		**00-00-2004 to 00-00-2004**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Efficient and Dynamic Response to Fire	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Postgraduate School,Department of Computer Science,Moves Institute,Monterey,CA,93943**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
Proceedings of the AAAI Workshop on Challenges in Game AI, 2004

14. ABSTRACT
Most first-person shooter game AI's are poor at quickly getting out of lines of fire. AI agents that pass up obvious opportunities to hide or take cover can ruin a game's immersiveness. We will present the sensor grid algorithm that can be used to address this problem, and has been implemented on top of America's Army. The algorithm performs a focused run-time search in the immediate vicinity of the agent. This allows it to be both fast and to react to changes in the environment.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**6**	

place to quickly duck out of fire is a waypoint is also small. To see why this is so, consider a map consisting of an open plane with one tall rock sitting in the center. By appropriately placing the observer and the hiding agent, one can make virtually any point the nearest place to hide! An additional difficulty, and one that will become more important in the future, is that a sparse set of potential hiding places fixed in advance is especially vulnerable to becoming invalid in dynamic environments because of vehicle motion, destruction of buildings, and creation of new hiding places such as piles of rubble, to name a few examples. Thus waypoint-based techniques typically result in agents that can behave very counter-intuitively when searching for cover.

In military simulations, space is typically represented by a fine rectangular grid (Reece 2003) (Reece 2000) (Richbourg and Olson 1996). This avoids the difficulties caused by a sparse spatial representation as described above, but at the cost of computational complexity that may be beyond the budget of many games. The memory required to store the grid may also be an issue for very constrained computational platforms, like game consoles.

Sensor Grid Overview

The sensor grid approach differs from its predecessors in that the set of possible hiding places is not fixed, but is instead generated dynamically at run-time. This allows it to be relatively dense close to the agent and sparse further out, while keeping the total size of the set small. Thus, this approach has the potential to provide some of the benefit of
a large set of potential hiding places while avoiding the computational complexity. Additionally, this approach mirrors the fact that humans can generally perceive nearby opportunities to hide more easily than ones in the distance, and furthermore, the nearer ones are more likely to be useful.

The sensor grid approach takes its name from the fact that the set of potential hiding places that are tested by the algorithm is fixed relative to the agent. It is as if the agent had a collection of observer-detecting sensors fixed with regard to the agent and one another moving wherever the agent moves. A simplified overview of the algorithm is provided in Figure 1. A complete description is given in the next section.

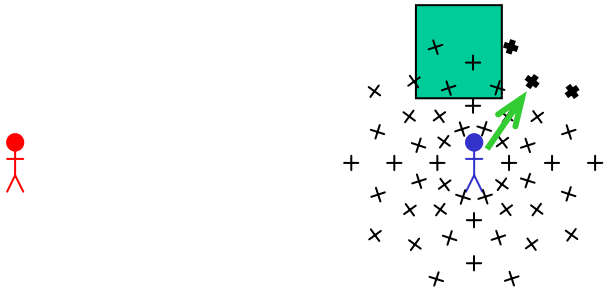

Figure 1: Top-down diagram illustrating the sensor grid approach. The agent (blue) is at right and a single observer (red) is at left. The array of senors (plus signs) surrounds the agent. A single vision-obstructing object is present (the green square). If a sensor cannot see the enemy, its location is hidden (bold plus signs). The agent chooses the nearest hidden sensor that is accessible (e.g. not inside an object),and moves there (green arrow).

Algorithmic Details

We now step through the sensor grid algorithm in detail. Details specific to the implementation of America's Army are left for the next section.

1. Find observers. The exact position of all observers (e.g. hostiles) is accessed.

2. Compute sensor locations. A sensor is chosen for processing. We use a sensor grid consisting of five staggered concentric rings of ten sensors each plus one at the agent's current position (in case it can hide simply by adjusting its posture), for a total of 51 sensors. The sensors closest to the agent are about 1.5 meters apart, while those furthest from the agent are about 3 meters apart. The absolute location of each sensor in the horizontal plane is determined by adding the sensor's position relative to the agent to the agent's absolute position.

3. Find ground level. The next step is to determine where ground level is at each sensor's horizontal location. This part is tricky and error-prone, since a vertical line at the sensor's horizontal position may intersect multiple objects from the environment. We limit our search with two biases. First, we limit our search to hiding places that are roughly at the same height as the agent. Second, since it is generally easier to move down then up, we prefer hiding places that are at the agent's height or below.

In the vertical plane containing the agent's location and the sensor, two 45 degree cones are constructed with the agent's location at the apex. The interior of the cones is considered close enough to the agent's height, and this "acceptable region" (see Figure 2) is augmented close to the agent to allow him to jump down a distance not exceeding its own height. At the

sensor's position and starting from the agent's height, a ray is traced downward until it strikes either a polygon or the edge of the acceptable region. If it strikes a polygon, this height is taken to be ground level. If not, the downward trace is repeated starting this time from the upper edge of the acceptable region. If both traces fail to strike a polygon, the corresponding sensor is eliminated from further consideration.

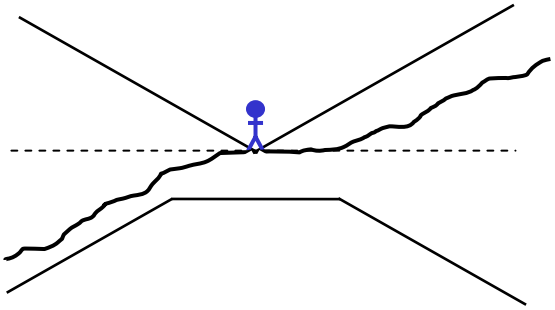

Figure 2: Diagram illustrating the edges of the "acceptable region" for finding ground level. The blue figure represents the agent's position. The heavy line represents a simple terrain mesh (though the region is defined and processed identically inside buildings, etc.). The straight lines represent the boundaries of the region. Note that to the left of the agent, ground level would be found on the first downward trace, but to the far right, ground level would be found only on the second trace.

4. Determine standability. Each sensor location is now tested for "standability", i.e. whether the agent can stand there without sliding away. A particular worry is that the location may be on the side of a steep cliff, for example. We avoid cases like this by requiring that the surface be within 45 degrees of horizontal. If the surface is not standable at a given sensor, it is eliminated from further consideration.

5. Determine visibility. We now check whether each sensor location is hidden from the observers. This is done by tracing lines of sight from the observers to various points on where the agent's body would be, were it located at the sensor location in one of the postures that the agent can adopt, e.g. "prone", "crouching", or "standing". For each posture, we check the visibility of three points at head height: one precisely at the position of sensor, and two displaced one collision radius to either side (see Figure 3).

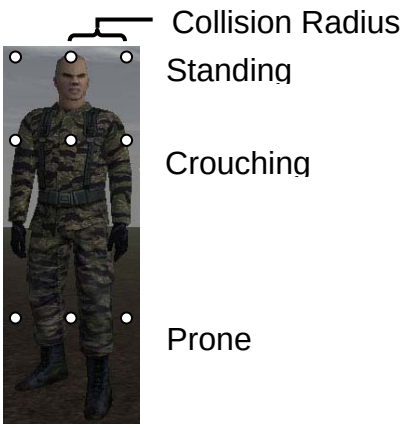

Figure 3: The nine points tested to determine visibility at a candidate hiding position. An alternative choice could include dropping the points to either side of the median line to the position of the shoulders in the corresponding posture.

Starting with the lowest posture, the visibility of all three points to each observer is checked. If any point is visible to any observer, the sensor is eliminated from further consideration. If all points at a particular posture are hidden, we proceed to the next higher posture, keeping note of the highest posture that is hidden.

6. Check accessibility. A good hiding place that we cannot quickly get to (e.g. inside a solid object, on the other side of a long wall) is not useful for ducking out of fire. We first check the straight-line accessibility of each sensor. This check determines if an upright cylinder of the same radius as the agent can move from the agent's current position to the sensor without intersecting any objects. If volumetric traces are not supported by the game engine, multiple ray traces, e.g. of the same sort used in step 5 can be used as an approximation. If the check fails, i.e. the straight-line path is obstructed, we then check whether there is some other straight-line accessible sensor, from which the current sensor is straight-line accessible. If a two-segment path is not found by this method, the sensor is removed from further consideration.

7. Move. The agent now selects the sensor whose path found in step 6 is the shortest. The agent moves to this sensor's location via this path. Upon arrival, it adjusts its posture to the highest posture that is hidden at this location, determined in step 5.

Unreal Implementation

The sensor grid approach was implemented on top of America's Army version 1.6, which uses Epic's Unreal Warfare engine. The code was written entirely in UnrealScript. The core of the code is small, about 500 lines in length including comments. The concealment testing and motion planning running on top of the standard game proved to be nearly instantaneous on

our tests on a laptop with 1.7 GHz Pentium IV and Geforce4 440 Go graphics card. Informal testing showed successful hiding in environments featuring various types of objects including as rocks, trees, vehicles, and buildings. The grid as presented is too coarse to guarantee the ability to navigate through doorways. An extension to the algorithm presented later (the inclusion of traditional waypoints in path generation) might help solve this problem. A screenshot from the demo is given as Figure 4.

Figure 4: The appearance of the agent inside the America's Army-based demo. The sensors are visualized as white glowing spheres at ground level.

Optimizations

Incremental Sensor Processing

Currently all sensors are processed for each step of the algorithm at once. However, since we want to go to the closest hiding place anyway, each sensor could potentially be processed independently starting from those near the agent. If there is a straight-line path to the candidate location, it could then be immediately accepted without processing all the other sensors.

Amortized Ray Casting

When multiple rays are cast to determine visibility, note that there is no absolute need to cast all these rays in one agent decision cycle. Instead, the most critical of the rays may be cast first, for example visibility of the top of the head, and an initial choice of concealed position may be made on that basis. In later decision cycles, additional rays can be cast, and the agent can re-plan its motion if it determines that its initial choice was bad.

Extensions

Improving Firing Positions

The sensor grid algorithm has offensive uses. We extended the agent to return fire after taking cover. The agent does this simply by adjusting his posture if possible. Otherwise, he moves back along whatever is between himself and the observer until the observer becomes visible and then commences firing.

Computing Cover Opportunities

"Cover" implies protection from fire while in the covered position, whereas "concealment" merely implies that one cannot be seen. With only very rare exceptions (e.g. bullet-proof glass), all covered positions are concealed, but not vice versa. When the assumption that some subset of environment polygons block fire is an acceptable approximation, the algorithm can be applied to compute positions that provide cover from direct fire, i.e. from projectiles that travel on trajectories that approximate straight lines. This is accomplished by running the algorithm on the set of fire-blocking polygons, instead of the set of opaque polygons.

When the algorithms are applied to the computation of cover, the assumption that fire is blocked by polygons is at best an approximation. Polygons are infinitely thin and cannot themselves block fire. A completely realistic computation of cover would therefore involve the details of the velocity and material of the projectile, the stopping power of the material, and the length over which the trajectory intersects the material. The algorithm described in this paper cannot be trivially extended to handle ballistics to this degree of accuracy. Note that this concern does not affect their application for the computation of concealment. Also, one might still consider using the algorithm presented here in a preprocessing step to determine promising candidates for a more accurate cover computation.

Integration with Waypoints

While the sensor grid was motivated as a replacement for navigating to safety on a sparse waypoint graph alone, a good set of waypoints can improve the performance of the sensor grid. The simplest approach is simply to extend the search for paths (step 6 above) to sensor locations to include any nearby waypoints. It might also be worthwhile not to count path segments generated using waypoints against the maximum allowed (two, as we have presented the algorithm). This trick might be particularly useful in improving the performance of the algorithm inside buildings and similarly constrained environments.

Large Numbers of Observers

Generally, there will be more than one observer for the algorithms to deal with. When the number of

observers is sufficiently small and their positions are known, we can represent them by the list of locations of their eyes. When there is a large number of observers or we only have a rough estimate of their position, one possible approach is to select a small set of representative viewpoints to deal with, for example, two viewpoints on the edge of the observer-occupied region of the space and one in the middle.

Partially-Transparent Polygons

Typically, the ray traces provided by a game engine assume that polygons in the environment are blocking or non-blocking in their entirety. But a traditional trick is to model objects with a lacy, open structure (e.g. a tree branch) as a single polygon overlaid with a partially-transparent (alpha) texture. Computing whether the precise location where a ray intersects a texture is transparent or opaque is possible, but computationally expensive. As environmental models become larger and involve greater numbers of smaller polygons, there is a trend towards modeling even fine structures with polyhedra instead of transparent textures. This problem should therefore become less and less significant over time.

Conclusions

We have presented the sensor grid algorithm, an approach to finding nearby hiding places that is efficient and robust against changes to the environment that occur during play. The algorithm is fairly straightforward to implement, and it seems to generally find good places to hide.

Like most related techniques, the sensor grid algorithm sometimes makes mistakes. These mistakes can be in either direction. If the points checked for visibility (Figure 3) are occluded, but most of the rest of the agent is not, the algorithm will falsely believe that the agent is hidden. Conversely, if a checked points are visible, but very little of the agent is showing (imagine an unlucky position behind some foliage), then the algorithm will believe a location exposed that is, at least for practical purposes, actually hidden. We have previously discussed other algorithmic approaches to hiding that are less prone to error, but are more complex both in terms of implementation and in terms of consuming computation cycles (Morgan 2003) (Darken 2004).

Computing lines of sight is already a major component of the computational budget devoted to AI for many computer games. Nonetheless, we believe that further investment in perceptual modeling, together with careful algorithm optimization, can yield large dividends of compelling behavior.

Acknowledgements

This work was supported by funds from the Naval Postgraduate School, the Navy Modeling and Simulation Management Office, and the U.S. Army Training and Doctrine Analysis Center, Monterey.

References

Darken, C. 2004. "Visibility and Concealment Algorithms for 3D Simulations", *Proceedings of Behavior Representation in Modeling and Simulation (BRIMS) 2004.*

Kelly, J., Beall, A., and Loomis, J. To appear. "Perception of Shared Visual Space: Establishing Common Ground in Real and Virtual Environments", to appear in *Presence.*

Morgan, D. 2003. "Algorithmic Approaches to Finding Cover in Three-Dimensional Virtual Environments", Master's Thesis, Naval Postgraduate School. Available at http://www.movesinstitute.org

Rabin, S. 2000. "A* Speed Optimizations", *Game Programming Gems*, Charles River Media, pp. 272—287.

Reece, D., Dumanoir, P. 2000. "Tactical Movement Planning for Individual Combatants", Proceedings of the 9th Conference on Computer Generated Forces and Behavioral Representation. Available at http://www.sisostds.org.

Reece, D. 2003. "Movement Behavior for Soldier Agents on a Virtual Battlefield." *Presence*, Vol. 12, No. 4, pp. 387—410, August 2003.

Richbourg, R., and Olson, W. 1996. "A Hybrid Expert System that Combines Technologies to Address the Problem of Military Terrain Analysis," *Expert Systems with Applications*, Vol. 11, No. 2, pp. 207—225.

Snook, G. 2000. "Simplified 3D Movement and Pathfinding Using Navigation Meshes", *Game Programming Gems,* Charles River Media, pp. 288—304.

Stout, B. 2000. "The Basics of A* for Path-Planning", *Game Programming Gems*, Charles River Media, pp. 254—263.

van der Sterren, W. 2002. "Tactical Path-Finding with A*", *Game Programming Gems 3*, Charles River Media, pp. 294—306.